13675

Network Working Group
Request For Comments # 435
NIC # 13675
Categories: TELNET, Protocols, Echoing
References: 318, 357

B. Cosell, BBN-NET
D. Walden, BBN-NET
5 January 1973

RCVD 15JAN73

TELNET Issues

This RFC discusses a number of TELNET related issues which have been bothering us*. The basic, central issue we started from was that of echoing. We worked downward from our difficulties to discover the basic principles at the root of our unhappiness, and from there worked back upwards to design a scheme which we believe to be better. In this note we will discuss both the alternate scheme and its underlying principles.

As something of a non sequitur, before discussing echoing we feel it expedient to dismiss one possible stumbling block, outright. HIDE YOUR INPUT may or may not be a good idea, this question not concerning us at the moment. Whatever the case, the issue of hiding input is certainly separable from that of echoing. We, therefore, strongly recommend that a STOP HIDING YOUR INPUT command be sanctioned to replace the multiplexing of this function on the NO ECHO command. Once this has been done, the pair of commands HIDE YOUR INPUT and STOP HIDING YOUR INPUT can be kept or discarded together, and we can discuss the issue of echoing independently of them.

---

* We have assumed that TELNET is defined as suggested by Jon Postel in RFC 318.

Echoing

The basic observation that we made regarding echoing was that servers seem to be optimized to best handle terminals which either do their own echoing or do not, but not both. Therefore, the present TELNET echoing conventions, which prohibit the server from initiating a change in echo mode, seemed overly confining. The servers are burdened with users who are in the 'wrong' mode, in which they might not otherwise have to be, and users, both human and machine, are burdened with remembering the proper echoing mode, and explicitly setting it up, for all the different servers. It is our understanding that this prohibition was imposed on the servers to prevent loops from developing because of races which can arise when the server and user both try to set up an echo mode simultaneously. We will describe a method wherein both parties can initiate a change of echo mode and show that the method does not loop.

This alternate specification relies on three primary assumptions. First as above, the server, as well as the user, should be able to suggest the echo mode. Second, all terminals must be able to provide their own echoes, either internally or by means of the local Host. Third, all servers must be able to operate in a mode which assumes that a remote terminal is providing its own echoes. Both of these last two result from the quest for a universal, miminal basis upon which to build. It is fairly easy for a Host which normally supplies echoes to disable the appropriate code, but it will difficult for a Host which

does not do echoing to integrate such routines into its system; similarly, it is easier for a local Host to supply echoes to a terminal which cannot provide its own, but it borders on the impossible to undo echoing in a terminal which has automatic echoing built into it.

Our proposed specification would use the present ECHO and NO ECHO commands as follows: ECHO, when sent by the server to the user, would mean 'I'll echo to you'; ECHO, when sent by the user to the server, would mean 'You echo to me'. NO ECHO, when sent by the server to the user, would mean 'I'll not echo to you'; NO ECHO, when sent by the user to the server, would mean 'Don't you echo to me'. These are, of course, nearly the same meanings that the commands currently have, although most current implementations seem to invert the server-to-user meanings.

In our specification, whenever a connection is opened both server and user assume that the user is echoing locally. If the user would, in fact, prefer the server to echo, the user could send off an ECHO command. Similarly, if the server prefers to do the echoing (for instance, because the server system is optimized for very interactive echoing), the server could send off an ECHO command. Neither is required to do anything, it is only a matter of preference. Upon receipt of either command by either party, if that is an admissable mode of operation the recipient should begin operating in that mode, and if such operation reflects a change in mode, it should respond with the same command to confirm that (and when) the changeover took

place. If the received command requests an inadmissable mode of operation, then the command's inverse should be sent as a refusal (this must be NO ECHO, since neither party can refuse a change into NO ECHO). To state these rules more formally:

1) Both server and user assume that a connection is initially in NO ECHO mode.

2) Neither party can refuse a request to change into NO ECHO mode.

3) Either party may send an unsolicited command only to request a change in mode.

4) A party only changes its echo mode when it receives an admissable request.

5) When a command is received, the party replies with its echo mode, unless it did not have to change mode to honor the request.

Several properties of this scheme are worthy of note:

1) NO ECHO is retained as the nominal connection mode. A connection will work in ECHO mode only when both parties agree to operate that way.

2) The procedure cannot loop. Regardless of which party (or both) initiates a change, or in what time order, there are at most three commands sent between the parties.*

3) Servers are free to specify their preferred mode of operation. Thus, human, or machine, users do not have to learn the proper mode for each server.

Three Principles

Let us mention the general principles we alluded to at the beginning of this note. The principles are: default

---

* Notice that a faulty implementation could achieve the effect of a loop by repeatedly sending a command which has previously been refused. We consider this a property of the implementation, not of the scheme; in general, a command which has be rejected should not be repeated until something changes -- for instance, not until after a different program has been started up.

implementation, negotiated options and symmetry. The principle of default implementation merely states that for all options, defaults are declared which must be implemented. It is this principle which leads us to seek out the 'minimum' for each option (to keep the required burden on everybody as small as possible), and prevents loops in the protocol. The principle of negotiated options merely states that options must be agreed upon by all (both) parties concerned. It is this principle which dictated the positive/negative acknowledgment scheme. The principle of symmetry merely states that neither party should have to 'know' whether it is the server or the user. Our scheme, as described thus far, is not totally symmetrical; we will consider this matter in a later section.

The Echoing scheme we have described, together with the principles stated above, form the heart of our comments on the TELNET protocol. The remainder of this note consists of further ways in which the protocol can be expanded; on the whole, these suggestions are all really only applications and development of the principles we have already put forward. However, the fecundity of these expansions, and the 'good feel' they have, make us yet more convinced of the 'rightness' of our original proposals.

Thus far, we have made a simple, concrete suggestion that we believe should be immediately sanctioned. Looking beyond that proposal, however, has suggestion a large number of further, more ambitious changes. The remainder of this RFC describes ideas which we don't feel have the immediacy of the proposal above, but should, nonetheless, be kept in mind if the network community decides to embark on revamping the protocol.

Synchronization

One complaint we have heard about the present convention for establishing an echoing mode is about the lack of a provision to synchronize a change of echoing mode with the user-to-server data stream; our scheme, too, is guilty on this count. John Davidson of the University of Hawaii has documented, in RFC 357, a more elaborate echoing scheme which doesn't have this problem. We, however, feel that it is possible to eliminate most of the trouble involved with normal changing of echo mode at a more modest cost than that required by the highly interactive scheme described by Davidson. We can do this by borrowing a small piece of that scheme. The rule we would incorporate is that whenever a party initiates a request for a change in echo mode, it then buffers, without transmitting or processing, all data in the user-to-server data stream until it receives an acknowledgment, positive or negative, at which time it deals with the buffered data in the newly negotiated mode. Since with both our proposed and the current schemes such a

request is guaranteed to be acknowledged, the buffering time is bounded.

An important aspect of this technique of eliminating the synchronization problem is that it need not ever become part of the official protocol. Since its operation is entirely internal to the server or user, each may independently weigh the value of elegance against the cost of the required code and buffer space.

Other Options

Abhay Bhushan has suggested to us that whether the user and server operate in line-at-a-time or character-at-a-time mode (see RFC 318) should also be a negotiated option. Further, he suggested that whether the terminal follows the TELNET end-of-line convention or not should also be negotiated. Thus, when a connection is opened, in addition to being set to NO ECHO mode, the terminal would also be set to LINE-AT-A-TIME and EOL modes. We could augment the command space with the new commands LINE, NO LINE (=CHARACTER), EOL and NO EOL (=separate CR and LF).

Once started in this direction, we found several further applications. HIDE YOUR INPUT could be made an option, as could Davidson's echoing scheme, and even the character set to be used! Consider that an APL subsystem might well want to suggest to its user that EBCDIC be used for the connnection.

In mentioning that the character set could be negotiated, it was implicit that 7-bit USASCII was the default. The possibility of having the default be straight binary suggests itself. If we augmented the protocol with a QUOTE character, the byte after which were to be always interpreted as data, then codes 128-255 could be retained as the 'TELNET command space' independently of the data mode in use by merely prefixing all data bytes in this region with a QUOTE. If BINARY were a permissible data mode, then it is easy to visualize many higher level protocols, e.g., perhaps, File Transfer and Graphics, being built on top of, and into, the TELNET protocol. What we would have accomplished is to promote TELNET from being a constrained, terminal-oriented protocol to its being a flexible, general protocol for any type of byte oriented communication. With such a backbone, many of the higher level protocols could be designed and implemented more quickly and less painfully -- conditions which would undoubtedly hasten their universal acceptance and availability.*

Looking toward a better world of the future, we have come up with a more compact and flexible command scheme. We'll describe it after the next section.

---

* Will Crowther, with an eye towards building higher protocols upon TELNET, has suggested that a SYNC command (not to be confused with the existing SYNCH), and a SYNC REPLY be added to TELNET. For example, a server might want to wait until the output buffer of a user's terminal were empty before doing something like closing the connection or passing the connection to another server. Although we see no current use for the command pair, they seem to be a handy enough building block that we recommend that they be included.

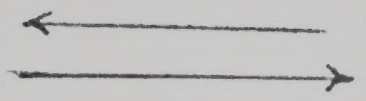
Symmetry

Some of the TENEX group (in particular, Thomas, Burchfiel and Tomlinson) have pointed out to us that although we have made the rules for the protocol symmetrical, we have not made the meanings of the commands symmetrical. For example, the interpretations of the ECHO command -- 'I'll echo to you' and 'You echo to me' -- implicitly assume that both the server and user know who is which. This is a problem not only for server-server connections, where it is not clear which is the user, but also for user-user connections, e.g., in linking Teletypes together, where it is not clear which is the server.

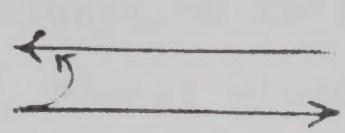
Responding to this, we came to understand that there are only five reasonable modes of operation for the echoing on a connection pair*:

---

* It is perhaps appropriate to mention that most of the connections in the network are TELNET connections, which are full duplex. Wouldn't it be reasonable to make all Host/Host protocol connections full duplex, rather than simplex? If, for some reason, one truly needs a simplex connection, the reverse direction can always just be ignored.

A  Process 1 Process 2

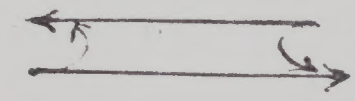
neither end echoes

B  Process 1 Process 2

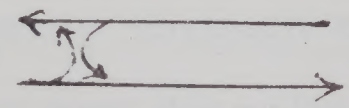
one end echoes for itself

C Process 1 Process 2

one end echoes for the other

D  Process 1 Process 2

both ends echo for themselves

E  Process 1 Process 2

one end echoes for both ends

The TENEX group suggested to us that four commands are sufficient to deal with completely symmetric echoing. We have actually already mentioned the four commands -- the two possible meainings for each of ECHO and NO ECHO. Explicitly, the commands would be I'LL ECHO TO YOU, YOU ECHO TO ME, DON'T ECHO TO ME and I'LL NOT ECHO TO YOU. Echoing is now the negotiation of two options, and the initial, default modes are DON'T ECHO TO ME and I'LL NOT ECHO TO YOU.

In the case where the server or user knows which he is, the modification to the scheme is minimal since the commands never had ambiguous meanings in these cases. When an 'end' truly doesn't know, then things are a little more complicated -- for example, consider both ends in I'LL ECHO TO YOU mode, but even then the problems are not

insurmountable.

Once the principle of symmetry is adopted, it is no longer possible to use a function in two different ways. On pages 5 and 6 of RFC 318, Postel gives a description of INS and SYNCH which indicates that they are used to simulate a 'break' user-to-server, but flush the output buffers server-to-user. Since we do believe in symmetry, we suggest that the INS/DATA-MARK be treated the same in both directions and that a new CLEAR YOUR BUFFER option be added.

Command Format

Extending full symmetry through the other options we have suggested, we can now describe the compacted command format referred to earlier.

Rather than having four commands for each option (I WILL, I WON'T, YOU DO, YOU DON'T), there would be four 'prefixes' -- WILL, WON'T, DO, DON'T -- which would be used before the single command devoted to each option, WON'T and DON'T being the default modes. To give an example, assume the codes for WILL and WON'T are 140 and 141, and the codes for ECHO REMOTE and HIDE INPUT are 132 and 133. Then several of the possible command combinations would be:

```
140 133  --  DO HIDE INPUT
140 132  --  DO ECHO REMOTE
141 132  --  WON'T ECHO REMOTE
141 133  --  WON'T HIDE INPUT
```

These are some of the commands that we believe should exist:

```
I WILL (140)
I WILL NOT (141)
YOU DO (142)
```

YOU DO NOT (143)
QUOTE (144)
SYNC (163)
SYNC REPLY (164)

ECHO REMOTE (132)
SEND A CHARACTER-AT-A-TIME (146)
SEND INDEPENDENT CR and LF (147)
SEND IN EBCDIC (162)
HIDE INPUT (133)
USE DAVIDSON'S ECHOING STRATEGY (145)

An important virtue of this command structure, and of our entire viewpoint, is that Hosts need no longer even be aware of what all the options are. If we call the mode of operation in which every alternative is in its default state the 'NVT', then a site, of course, must handle an NVT, but beyond that if it merely responds no to any command it does not understand, then it can totally ignore options it chooses not to implement. Thus, options would truly be optional (for a change), not only to the user who may choose not to invoke them, but also to the systems builders who may now choose not to offer them!

We hereby volunteer to rigorously specify a version of TELNET which embodies the principles we have described and to do so at any level of complexity deemed sufficient by the network community.

Appendix: A Sample Implementation

The basic scheme we described represents most of what we have been thinking about; the further extensions are just that, extensions. We fear, however, that some who are spiritually in league with us might be frightened off by the magnitude of all the changes we suggest. To combat this, we here provide an example of how simply and straight-forwardly the basic scheme could be implemented for the TIP*.

For each user terminal the TIP would keep three state bits: whether the terminal echoes for itself (NO ECHO always) or not (ECHO mode possible), whether the (human) user prefers to operate in ECHO or NO ECHO mode, and whether the connection to this terminal is in ECHO or NO ECHO mode. We call these three bits P(hysical), D(esired) and A(ctual).

When a terminal dials up the TIP, the P-bit is set appropriately, the D-bit is set equal to it, and the A-bit is set to NO ECHO. The P- and A-bits may be manually reset by direct commands if the user so desires; for instance, a user in Hawaii on a 'full-duplex' terminal might know that whatever the preference of a mainland server, because of satellite delay his terminal had better operate in NO ECHO mode -- he would direct the TIP to change his D-bit from ECHO to NO ECHO.

When a connection is opened from the TIP terminal to a server, the TIP would send the server an ECHO command if the

---

* Readers unfamiliar with the TIP may read the TIP Users Guide -- NIC 10916.

MIN (with NO ECHO less than ECHO) of the P- and D-bits is different from the A-bit. If a NO ECHO or ECHO arrives from the server, the TIP will set the A-bit to the MIN of the received request, the P-bit and the D-bit. If this changes the state of the A-bit, it will send off the appropriate acknowledgment; if it does not, then the TIP will send off the appropriate refusal if not changing meant that it had to deny the request (i.e., the MIN of the P- and D- bits was less than the received A- request). If while a connection is open, the TIP terminal user changes either the P- or D-bit, the TIP will repeat the above tests and send off an ECHO or NO ECHO, if necessary. When the connection is closed, the TIP would reset the A-bit to NO ECHO.

While the TIP's implementation would not involve ECHO or NO ECHO commands being sent to the server except when the connection is opened or the user explicitly changes his echoing mode, we would suppose that bigger Hosts might send these commands quite frequently. For instance, if a JOSS subsystem were running, the server might put the user in NO ECHO mode, but while DDT was running, the server might put the user in ECHO mode.